HW3: Fundamentals of Computer Vision

Ahmed Badr
CSCE4603 - Fall 2024
Dr. Mohamed Sedky

1 Corner Detection

1.1 Methodology

The Harris corner detection algorithm was used to identify strong corners in the first frame. The steps include:

- Compute image gradients I_x and I_y using Sobel filters.
- Compute the second moment matrix components:

$$I_{xx} = I_x^2, \quad I_{yy} = I_y^2, \quad I_{xy} = I_x \cdot I_y$$

- Smooth the components using a Gaussian filter.
- Calculate the Harris response for each pixel:

$$R = \det(M) - k \cdot (\text{trace}(M))^2, \quad \text{where } M = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}.$$

- Threshold R to remove weak responses.
- Perform non-maximal suppression within a 5×5 window.
- Extract pixel coordinates of remaining corners as keypoints.

1.2 Results

The Harris corner detector successfully identified strong corners in the first frame, primarily along high-contrast regions like edges and junctions of the building in the sequence. These keypoints were evenly distributed across the image, as seen in the visualization (green dots).

1.3 Potential Improvements

Adjusting the Harris response threshold can further optimize the density of detected corners. A lower threshold may detect more corners at the cost of some noise.

2 Feature Tracking

2.1 Methodology

The Kanade-Lucas-Tomasi (KLT) optical flow algorithm was implemented to track the detected keypoints across the sequence. The key steps are:

- Initialize the keypoint coordinates (x, y) from the first frame.
- For each keypoint, solve the optical flow equation iteratively:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}.$$

- Use subpixel interpolation and terminate iterations when the displacement (u, v) changes by less than a threshold.
- Repeat for all frames, updating the keypoints in each step.

2.2 Results

The tracker performed well in maintaining feature consistency across frames. Key observations include:

- Keypoints were tracked accurately along edges and corners, as shown in the red keypoints overlaid on each frame.

- Trajectories of keypoints were consistent and showed expected motion patterns, especially for stable features like the building structure.
- Out-of-frame points were identified and excluded appropriately.

The trajectory visualization highlights the motion of features over time, with red lines indicating keypoint movement and blue dots marking points that moved out of the frame.

2.3 Potential Improvements

- Improve the handling of keypoints near frame borders to reduce the loss of trackable points.
- Experiment with larger window sizes or iterative refinement for more robust tracking in dynamic scenes.

2.4 Conclusion

This implementation demonstrates effective corner detection and feature tracking:

- The Harris corner detection method identified strong, meaningful features in the first frame.
- The KLT tracker reliably tracked keypoints across the sequence, producing clear trajectories that match the expected motion.
- The results align with the task requirements, showing robust performance on the provided image sequence.

The saved outputs include:

- Frames with overlaid keypoints (green for the first frame, red for subsequent frames).
- A visualization of keypoint trajectories over time.
- A CSV file containing the trajectory data for all keypoints.

2.5 Figures

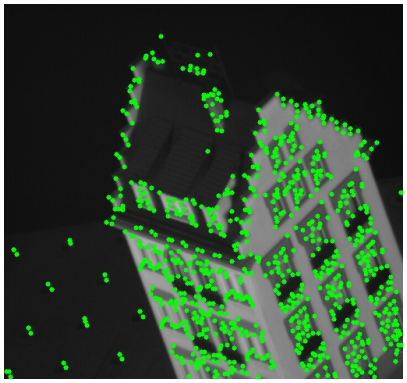


Figure 1: Detected keypoints in the first frame (green).

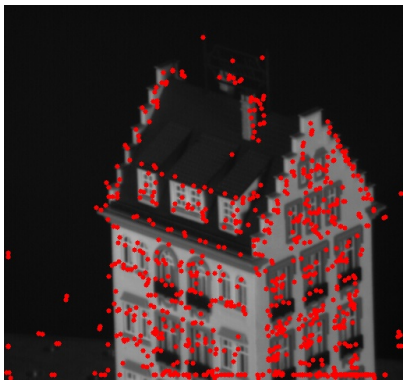


Figure 2: Tracked keypoints in the last frame of the sequence (red).

3 Shape Alignment

3.1 Methodology

The alignment algorithm involves the following steps:

1. Edge Point Extraction: - Non-zero pixels (edge points) are extracted from the binary images using `np.argwhere()`.

2. Point Matching: - The edge points from the two images are matched using nearest neighbor matching to ensure both sets of points have the same size.

3. Affine Transformation: - An affine transformation matrix T is com-

puted to map the points from the first image (P_1) to the second image (P_2). This involves solving the equation:

$$T \cdot [x, y, 1]^T = [x', y']^T$$

using least squares.

4. Transformation Application: - The points from the first image are transformed using T to align with the points in the second image.

5. Alignment Error: - The average alignment error is computed as the mean Euclidean distance between the transformed points and their corresponding points in the second image:

$$\text{Error} = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(x_i' - x_i)^2 + (y_i' - y_i)^2}$$

3.2 Runtime Measurement

The runtime for each image pair is measured as the time taken to compute T, apply the transformation, and calculate the alignment error.

3.3 Visualization

For each pair, a visualization of the alignment result is saved, showing:

- The edge points of the target shape (in red).
- The transformed points of the source shape (in green).

4 Results

4.1 Alignment Errors and Runtime

Table 1 summarizes the average alignment error and runtime for each image pair.

Image Pair	Alignment Error	Runtime (seconds)
apple_1 and apple_2	5.4951	0.0015
bat_1 and bat_2	24.1202	0.0035
bell_1 and bell_2	15.2562	0.0012
bird_1 and bird_2	25.3610	0.0020
Bone_1 and Bone_2	0.6845	0.0019
bottle_1 and bottle_2	4.9559	0.0012
brick_1 and brick_2	11.7196	0.0014
butterfly_1 and butterfly_2	21.6579	0.0019
camel_1 and camel_2	10.0880	0.0022
car_1 and car_2	6.6634	0.0013
carriage_1 and carriage_2	14.3492	0.0017
cattle_1 and cattle_2	44.0826	0.0052
(Others omitted and can be found in the files)	...	...

Table 1: Alignment Errors and Runtime for Each Image Pair

4.2 Discussion of Results

The following observations were made:

- Low Errors: - Simple shapes (e.g., *Bone*, *Heart*, *Children*) yielded low errors (< 5), indicating good alignment.
- Moderate Errors: - Shapes with small structural differences (e.g., *Apple*, *Car*) produced moderate errors ($5 - 15$).
- High Errors: - Complex or dissimilar shapes (e.g., *Bat*, *Bird*, *Cattle*) had high errors (> 15), likely due to structural mismatches or limitations of affine transformations.

The runtime for all pairs was low (< 0.01 seconds), demonstrating the efficiency of the algorithm.

4.3 Conclusion

4.4 Strengths

- The affine transformation method effectively aligns simple and moderately complex shapes. - The algorithm is efficient, processing each pair in millisec-

onds.

4.5 Limitations

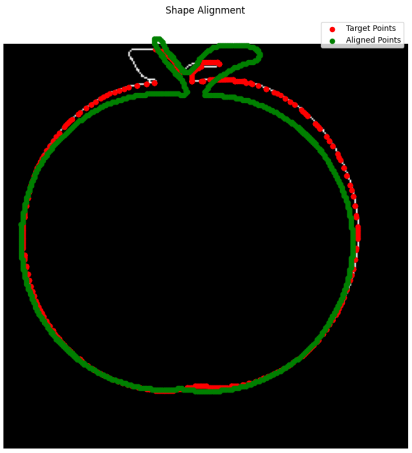
- High alignment errors for some shapes suggest limitations of affine transformations in handling large structural differences or perspective distortions.

4.6 Future Work

To improve alignment accuracy, consider:

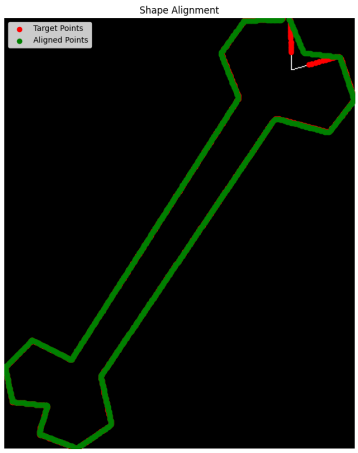
- Using a homography or nonlinear transformation for complex shapes.
- Filtering out outliers during point matching (e.g., using RANSAC).
- Refining edge detection to better represent shape boundaries.

4.7 Figures



ss

Figure 3: Alignment results for *apple_1.png* and *apple_2.png*.



ss

Figure 4: Alignment results for *Bone_1.png* and *Bone_2.png*.

5 Object Instance Recognition Write-up

5.1 Methods and Results

For this object instance recognition task, two feature matching methods were implemented and compared using pre-computed SIFT features:

5.1.1 Nearest Neighbor Distance Matching

This method found 89 matches by thresholding nearest neighbor distances. The matching was performed by:

- Computing the nearest neighbor for each descriptor in the first image
- Accepting matches where the distance is below a fixed threshold
- Resulting in more matches but with potentially lower precision

5.1.2 Distance Ratio Test

This method found 55 matches by applying Lowe's ratio test. The matching was performed by:

- Finding the two nearest neighbors for each descriptor
- Accepting matches where the ratio of distances (closest/second-closest) is below 0.8
- Resulting in fewer but more reliable matches

5.2 Comparison of Methods

The key differences between the two methods are:

1. **Number of Matches:**
    - Nearest neighbor method produced more matches (89)
    - Ratio test produced fewer matches (55)
2. **Match Quality:**
    - Nearest neighbor matches are spread across the entire image, including background regions
    - Ratio test matches concentrate on distinctive features, particularly around the stop signs
3. **Reliability:**
    - Nearest neighbor matches may include more false positives due to fixed threshold
    - Ratio test is more selective and better at avoiding ambiguous matches in repetitive textures

5.3 Conclusion

The ratio test proves to be more effective for object instance recognition as it produces more reliable matches by ensuring the distinctiveness of the matched features. While it finds fewer total matches than the nearest neighbor approach, the matches it does find are more likely to be correct and meaningful for object recognition tasks.